DSA-II PRACTICAL SHORT NOTES (ANDY)

- **Binary Search Tree (BST) :**

```
#include<stdio.h>
#include<stdlib.h>
struct node
 {
   struct node *lchild;
   int data;
   struct node *rchild;
 };
typedef struct node NODE;

NODE *getnode()
 {
   NODE *temp;

temp=(NODE*)malloc(sizeof(NODE))
;
   printf("\n\n Enter the data : ");
   scanf("%d",&temp->data);
   temp->lchild=NULL;
   temp->rchild=NULL;
    return(temp);
}
NODE *create()
 {
   NODE *temp,*ptr,*root;
   char ch;
   root=NULL;
   do
                {
                 temp=getnode();
                 if(root==NULL)
                            root=temp;
                 else
                {
                 ptr=root;
                 while(ptr!=NULL)
                  {
                            if(temp-
>data<ptr->data)
                            {

                if(ptr->lchild==NULL)

                {

                            ptr-
>lchild=temp;

                            break;

                }

                else

                  ptr=ptr->lchild;
                            }
                            else
                            {

                if(ptr->rchild==NULL)

                {

                            ptr-
>rchild=temp;

                            break;
                            }

                            else

                             ptr=ptr->rchild;
                                        }
                             }//while
                            } //else
                             printf("\n Add More
(Y/N)? : ");
      scanf(" %c",&ch);
   }while(ch=='Y' || ch=='y');
  return(root);
 }
int search(int num,NODE *ptr)
{
   while(ptr!=NULL)
                  {
                     if(num==ptr->data)
                      return 1;

if(num<ptr->data)
                                        {

                            ptr=ptr->lchild;

                                        }

if(num>ptr->data)
                                        {

                             ptr=ptr->rchild;
                                        }



                             }

  return 0;



}
NODE *insert(NODE *temp, NODE
*root)
{
  NODE *ptr;
  ptr=root;
  if(ptr==NULL)
   {
     root=ptr=temp;
   }
  else
   {
     ptr=root;
     while(ptr!=NULL)
                 {
                  if(temp->data<ptr-
>data)
                   {
                    if(ptr-
>lchild==NULL)
                     {
                                       ptr-
>lchild=temp;
                                       break;
                     }
                    else
                                       ptr=ptr-
>lchild;
                   }
                  else
                   {
                                       if(ptr-
>rchild==NULL)
                                        {
                                                ptr-
>rchild=temp;
                                                break;
                                               }
                                               else
                                                ptr=ptr-
>rchild;
                                   }
                                }
   return(root);
  }

return(root);
}



void inorder(NODE *ptr)
{
 if(ptr!=NULL)
 {
   inorder(ptr->lchild);
   printf(" %d",ptr->data);
   inorder(ptr->rchild);
 }
}



void preorder(NODE *ptr)
{
 if(ptr!=NULL)
 {
   printf(" %d",ptr->data);
   preorder(ptr->lchild);
   preorder(ptr->rchild);
 }
}
void postorder(NODE *ptr)
{
 if(ptr!=NULL)
 {
   postorder(ptr->lchild);
   postorder(ptr->rchild);
   printf(" %d",ptr->data);
 }
}

main()
 {
   int ch,num,t,abc;
   NODE *root;
   NODE *temp;
   while(1)
   {
   printf("\nMain Menu");
   printf("\n1: Create Binary search
tree");
   printf("\n2: Inorder traversal");
   printf("\n3: Preorder traversal");
   printf("\n4: postorder traversal");
   printf("\n5: Search a value");
   printf("\n6: Insert a value");
   printf("\n7: Exit");
   printf("\n Enter the choice: ");
   scanf("%d",&ch);
   switch(ch)
   {
    case 1:   root=create();
                                        break;
    case 2:

printf("\nInorder traversal");

inorder(root);
                                        break;
    case 3:
printf("\nPreorder traversal:  ");

preorder(root);
                                  break;
      case 4:

printf("\nPostorder Traversal: ");

postorder(root);
                                  break;
      case 5:   printf("\nEnter the Value
to be searched: ");

scanf("%d",&num);

t=search(num,root);
                                  if(t==1)

printf("\nValue is found");
                                  else

              printf("\nValue is not
found");
                                  break;
      case 6:

              temp=getnode();

              root=insert(temp,root);
                                 break;
      case 7:
                                 exit(1);
  }
 }
}
```

- **Adjacency Matrix with In degree , Out degree , Total degree :**

```
#include<stdio.h>
#include<stdlib.h>
struct NODE
{
  int data;
  struct NODE *next;
};
typedef struct NODE node;
node *list[10];
int nov;
node *getnodenum(int vno)
{
   node *temp;
   temp=(node*) malloc(sizeof(node));
   temp->data=vno;
   temp->next=NULL;
   return temp;
}
void display(node *list[10])
{
  int i;
  node *ptr;
  for(i=1;i<=nov;i++)
  {
                printf("V%d  ",i);

for(ptr=list[i];ptr!=NULL;ptr=ptr-
>next)

printf("%d->",ptr->data);
                printf("NULL");
                printf("\n");
  }
}
void creatadjacencylist()
{
            int i,j;
            char ch;
            node *temp,*last;
            for(i=1;i<=nov;i++)

            list[i]=NULL;
            for(i=1;i<=nov;i++)
            {

            for(j=1;j<=nov;j++)
                        {

            printf("\nIs there edge between V[%d] and V[%d] (choose : y/n): ",i,j);

            scanf(" %c",&ch);

            if(ch=='Y' || ch=='y')

            {


            temp=getnodenum(j);


            if(list[i]==NULL)


            list[i]=temp;

                        else

                        {


            for(last=list[i];last->next!=NULL;last=last->next);


            last->next=temp;

                        }

            }
                        }
            }


}
void degree()
{
   int i,cnt,outcnt[10],incnt[10]={0};
   node *ptr;
   for(i=1;i<=nov;i++)
    {

for(ptr=list[i],cnt=0;ptr!=NULL;ptr=ptr->next,cnt++)
        incnt[ptr->data]+=1;
      outcnt[i]=cnt;
    }
   printf("\nVertex\t   Indegree\t Outdegree\t  Totaldegree");
   for(i=1;i<=nov;i++)
     printf("\nV%d \t\t %d \t\t %d \t\t %d",i,incnt[i],outcnt[i],incnt[i]+outcnt[i]);
}
main()
{
  int i,j,a[20][20];
  printf("\nEnter no. of vertices: ");
  scanf("%d",&nov);

  creatadjacencylist();
  printf("\n******Adjacency List*******\n");
  display(list);
 degree();
}
```

- **Leaf Node :**

```
#include<stdio.h>
#include<stdlib.h>

int nodetotal=0,leaftotal=0;

struct node
{
            struct node *lchild;
            int data;
            struct node *rchild;
};
typedef struct node NODE;

NODE *getnode()
{
            NODE *temp;
            temp=(NODE*)malloc(sizeof(NODE));
            printf("\n\n Enter the data : ");
            scanf("%d",&temp->data);
            temp->lchild=NULL;
            temp->rchild=NULL;
            return(temp);
}

NODE *create()
{
            NODE *temp,*ptr,*root;
            char ch;
            root=NULL;
            do
            {

            temp=getnode();

            if(root==NULL)

            root=temp;
                        else
                        {

            ptr=root;

            while(ptr!=NULL)

            {

                        if(temp->data<ptr->data)

                        {

            if(ptr->lchild==NULL)

            {

                        ptr->lchild=temp;

                        break;

            }

            else

                        ptr=ptr->lchild;

                        }

                        else

                        {

            if(ptr->rchild==NULL)

            {

                        ptr->rchild=temp;

                        break;

            }

            else

                        ptr=ptr->rchild;

                        }

            }//while
                        } //else
                        printf("\n Add More (Y/N)? : ");
                        scanf(" %c",&ch);
            }while(ch=='Y' || ch=='y');
            return(root);
}

int totalnode(NODE *ptr)
{
            if(ptr!=NULL)
            {

            nodetotal++;

            totalnode(ptr->lchild);

            totalnode(ptr->rchild);
            }
            return nodetotal;
}
int leafcount(NODE *ptr)
{
            if(ptr!=NULL)
            {

            leafcount(ptr->lchild);

                        if(ptr->lchild==NULL && ptr->rchild==NULL)

            leaftotal++;

            leafcount(ptr->rchild);
            }
            return leaftotal;
}

main()
{
            NODE *root;
            root=create();
            printf("\n");
            nodetotal=totalnode(root);
            leaftotal=leafcount(root);
            printf("\n The total no. of nodes are : %d",nodetotal);
            printf("\n\n The total no. of leaf nodes are : %d",leaftotal);
}
```

- **Adjacency List :**

```
#include<stdio.h>
#include<stdlib.h>
struct NODE
{
  int data;
  struct NODE *next;
};
typedef struct NODE node;
node *list[10];
int nov;
node *getnodenum(int vno)
{
   node *temp;
   temp=(node*) malloc(sizeof(node));
   temp->data=vno;
   temp->next=NULL;
   return temp;
}
void display(node *list[10])
{
  int i;
  node *ptr;
  for(i=0;i<nov;i++)
   {
              printf("V%d  ",i+1);

for(ptr=list[i];ptr!=NULL;ptr=ptr->next)

printf("%d->",ptr->data);
              printf("NULL");
              printf("\n");
   }


}
void creatadjacencylist()
{
            int i,j;
            char ch;
            node *temp,*last;
            for(i=0;i<nov;i++)

            list[i]=NULL;
            for(i=0;i<nov;i++)
            {

            for(j=0;j<nov;j++)
                        {
```

```
                printf("\nIs there edge
between V[%d] and V[%d] (Choose
(y/n): ",i+1,j+1);

                scanf(" %c",&ch);

                if(ch=='Y' || ch=='y')

                {


                temp=getnodenum(j+1);


                if(list[i]==NULL)


                list[i]=temp;

                                else

                                {


                for(last=list[i];last-
>next!=NULL;last=last->next);


                last->next=temp;

                                }

                }
                                }
                }

                //display(list);
}

main()
{
  int i,j,a[20][20];

  printf("\nEnter no. of vertices: ");
  scanf("%d",&nov);

  creatadjacencylist();
  printf("\n******Adjacency
List*******\n");
  display(list);
}
```

- **BFS :**

```
#include<stdio.h>
#define MAX 20

struct n
 {
   int data;
   struct n *link;
 };

typedef struct n NODE;
NODE *h[20];
struct queue
{
  int front,rear;
  int Q[MAX];
};
typedef struct queue QUEUE;
QUEUE *q;

void initqueue()
{
  int i;
  for(i=0;i<MAX;i++)
    q->Q[i]=0;
  q->rear=-1;
  q->front=-1;
  printf("\nQueue created");
}
void add(int data)
{

   q->Q[++q->rear]=data;
}



int delet()
{

   return(q->Q[++q->front]);
}
int isempty()
{
  if (q->rear==q->front)
    return 1;
  else
   return 0;
}

void accept(int n,int a[MAX][MAX])
{
                int i,j;
                for(i=0;i<n;i++)
   {
    for(j=0;j<n;j++)
     {
                printf("\n Enter
a[%d][%d] : ",i+1,j+1);
                scanf("%d",&a[i][j]);
     }
   }
}
void display(int n,int a[MAX][MAX])
{
                int i,j;
                for(i=0;i<n;i++)
   {
    for(j=0;j<n;j++)
     {
                printf("\t %d ",a[i][j]);
     }
    printf("\n");
   }
}
void bfs(int n,int a[MAX][MAX])
{
  int i,j,v[10]={0};
  printf("\n \t BFS seq. :\n ");
  i=1;
  add(i);
  v[i-1]=1;
  while(isempty()!=1)
   {
                i=delet();
                for(j=0;j<n;j++)
                 {
                  if(a[i-1][j]==1 &&
v[j]==0)
                   {
                                add(j+1);
                                v[j]=1;
                   }
                 }
                printf("\t V%d ",i);
   }
}

NODE *getnode(int j)
 {
   NODE * temp;
   temp=(NODE
*)malloc(sizeof(NODE));
   temp->data=j;
   temp->link=NULL;
   return(temp);
 }

NODE *findlast(NODE *h)
 {
   NODE *ptr;
   for(ptr=h;ptr->link!=NULL;ptr=ptr-
>link);
   return(ptr);
 }

void displaylist(NODE *h[10],int n)
 {
   NODE *ptr;
   int i;
   for(i=0;i<n;i++)
    {
     printf("\n V%d ",i+1);
     ptr=h[i];
     if(ptr==NULL)
      printf(" NULL");
     while(ptr!=NULL)
      {
                printf(" -> %d",ptr-
>data);
                ptr=ptr->link;
      }
     printf("\n");
    }
 }
void adjmat(int n,int a[MAX][MAX])
{
  NODE *ptr,*temp;
  int i,j;
   for(i=0;i<n;i++)
    h[i]=NULL;
   for(i=0;i<n;i++)
    {
     for(j=0;j<n;j++)
      {
                if(a[i][j]!=0)
                 {
                  temp=getnode(j+1);
                  if(h[i]==NULL)
                   h[i]=temp;
                  else
                   {
                    ptr=findlast(h[i]);
                    ptr->link=temp;
                   }
                 }
      }
    }
   printf("\n The Adjacency list looks
like ... \n");
   displaylist(h,n);


}


void main()
 {
   int ver,matric[MAX][MAX];
                clrscr();
                printf("\n\t\t\t Enter the
no. of vertex ");
                scanf("%d",&ver);
                accept(ver,matric);
                printf("\nAdjacency
matrix:- \n");
                display(ver,matric);
                getch();
                adjmat(ver,matric);
                getch();
                bfs(ver,matric);
                getch();
}
```

- **DFS :**

```
#include<stdio.h>
int nov,a[20][20];
int visited[20];
void creatematrix()
{
                int i,j;
                printf("\nEnter no. of
vertices: ");
                scanf("%d",&nov);

                //accept the matrix
                for(i=1;i<=nov;i++)
                {

                for(j=1;j<=nov;j++)
                                {

                printf("\nIs there egde
between V[%d] and V[%d]: ",i,j);

                scanf("%d",&a[i][j]);
                                }
                }


}
void display(int a[20][20])  //print the
matrix
{
                int i,j;
                for(i=1;i<=nov;i++)
                {

                for(j=1;j<=nov;j++)

                printf("\t%d",a[i][j]);

                printf("\n");
                }
}
void recdfs(int a[20][20],int nov,int
ver)
{
                int i;
                visited[ver]=1;
                printf(" V%d",ver);
                for(i=1;i<=nov;i++)
                {


                if((a[ver][i]==1) &&
(visited[i]==0))



                recdfs(a,nov,i);
                                }


}
main()
{
                int ch,i;
                creatematrix();
                printf("\n\t***Adjacency
Matrix****\n");
```

```
                display(a);

        printf("\nThe Depth First search
Traversal(DFS) is:");
        recdfs(a,nov,1);
}
```

- **Heap Sort (by putting value in int main()) :**

```
#include <stdio.h>
void swap(int* a, int* b)
{
int temp = *a;
*a = *b;
*b = temp;
}
void heapify(int arr[], int N, int i)
{
int largest = i;
int left = 2 * i + 1;
int right = 2 * i + 2;
if (left < N && arr[left] > arr[largest])
largest = left;
if (right < N && arr[right] >
arr[largest])
largest = right;
if (largest != i)
 {
swap(&arr[i], &arr[largest]);
heapify(arr, N, largest);
}
}
void heapSort(int arr[], int N)
{
for (int i = N / 2 - 1; i >= 0; i--)
heapify(arr, N, i);
for (int i = N - 1; i >= 0; i--) {
swap(&arr[0], &arr[i]);
heapify(arr, i, 0);
}
}
void printArray(int arr[], int N)
{
for (int i = 0; i < N; i++)
printf("%d ", arr[i]);
printf("\n");
}
int main()
{
int arr[] =
{10,50,60,40,30,20,80,90,70,100};
int N = sizeof(arr) / sizeof(arr[0]);
heapSort(arr, N);
printf("Sorted array is\n");
printArray(arr, N);
}
```

- **BST sum odd() & sum even () :**

```
// C++ program to print all odd node of
BST
#include <bits/stdc++.h>
using namespace std;

// create Tree
struct Node {
                int key;
                struct Node *left, *right;
};

// A utility function to create a new
BST node
Node* newNode(int item)
{
                Node* temp = new Node;
                temp->key = item;
                temp->left = temp->right
= NULL;
                return temp;
}

// A utility function to do inorder
traversal of BST
void inorder(Node* root)
{
                if (root != NULL) {

                inorder(root->left);
                                printf("%d
", root->key);

                inorder(root->right);
                }
}

/* A utility function to insert a new
node
with given key in BST */
Node* insert(Node* node, int key)
{
                /* If the tree is empty,
return a new node */
                if (node == NULL)
                                return
newNode(key);

                /* Otherwise, recur down
the tree */
                if (key < node->key)
                                node->left
= insert(node->left, key);
                else
                                node->right
= insert(node->right, key);

                /* return the
(unchanged) node pointer */
                return node;
}

// Function to print all odd nodes
void oddNode(Node* root)
{
                if (root != NULL) {

                oddNode(root->left);

                                // if node is
odd then print it
                                if (root-
>key % 2 != 0)

                printf("%d ", root->key);


                oddNode(root->right);
                }
}

// Driver Code
int main()
{

                Node* root = NULL;
                root = insert(root, 5);
                root = insert(root, 3);
                root = insert(root, 2);
                root = insert(root, 4);
                root = insert(root, 7);
                root = insert(root, 6);
                root = insert(root, 8);

                oddNode(root);

                return 0;
}
```

Even

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int key;
   struct node *left,
*right;
};

struct node
*newNode(int item) {
struct node *temp=(struct
node
*)malloc(sizeof(struct
node));
   temp->key = item;
   temp->left = temp-
>right = NULL;
   return temp;
}

struct node* insert(struct
node* node, int key) {
   if (node == NULL)
return newNode(key);
   if (key < node->key)
      node->left =
insert(node->left, key);
   else
      node->right =
insert(node->right, key);
   return node;
}

void main(void)
{
   struct node *root =
NULL;
   int
data[]={3,1,2,6,4,7,8}, n,
i;

n=sizeof(data)/sizeof(dat
a[0]);
   for(i=0;i<n;i++)

root=insert(root,data[i]);
}
```

****